

DYNAMIC PROGRAMMING

การอบรมโอลิมปิกวิชาการและการพัฒนามาตรฐานวิทยาศาสตร์
และคณิตศาสตร์ สาขาคอมพิวเตอร์ ค่ายที่ 2
ศูนย์คณะวิทยาศาสตร์และเทคโนโลยี
มหาวิทยาลัยสงขลานครินทร์ วิทยาเขตปัตตานี

2

KNAPSACK PROBLEM

3

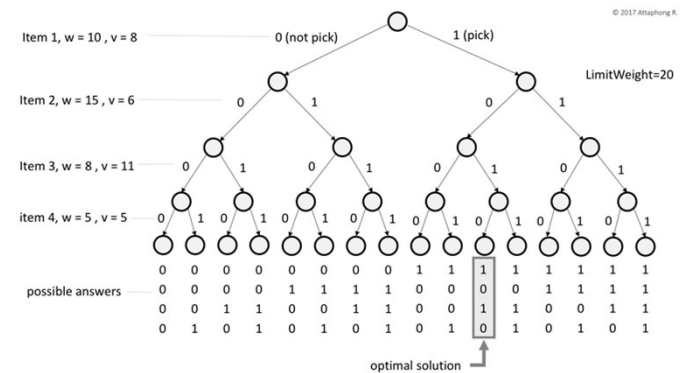
Knapsack Problem

- ปัญหาการหยิบของใส่ถุงเป้ โดยต้องเลือกหยิบของใส่ในถุงให้มีมูลค่ารวมสูงสุด แต่ห้ามใส่ของเกินน้ำหนักที่กำหนด โดยของแต่ละชิ้นจะมีน้ำหนักและมูลค่าที่ต่างกันออกไป
- ตัวอย่าง กำหนดให้ถุงเป้รับน้ำหนักได้ไม่เกิน 20

Item	1	2	3	4
Weight	10	15	8	5
Value	8	6	11	5

4

Knapsack Problem



Knapsack Problem

- The idea of dynamic programming
 - มักเป็นปัญหาเพื่อหาค่าเหมาะสมสุด (Optimal Solution) และสามารถหาความสัมพันธ์เวียนเกิด (Recurrence Relation) ได้
 - หาผลลัพธ์เริ่มต้นจากปัญหาย่อย แล้วนำคำตอบที่ได้ไปใช้หาผลลัพธ์ของปัญหาที่ใหญ่ขึ้น

Knapsack Problem

- กำหนดให้มีสิ่งของ n ชิ้น โดยแต่ละชิ้นจะมีน้ำหนักเป็น w_i และมีมูลค่าเป็น v_i หากเรามีถุงเป้ที่สามารถจุสิ่งของน้ำหนักไม่เกิน W เรา จะเลือกสิ่งของเหล่านี้ได้อย่างไรให้มีมูลค่ารวมสูงสุด โดยที่น้ำหนัก สิ่งของรวมไม่เกิน W

$$\begin{aligned} &\text{maximizes } \sum_{i \in T} v_i \\ &\text{subject to } \sum_{i \in T} w_i \leq W, T \subseteq \{1, 2, \dots, n\} \end{aligned}$$

Dynamic Programming for Knapsack Problem

- Define an optimal solution to recurrence relation
 - กำหนดให้ $V[i, w]$ แทนมูลค่ารวมสูงสุดของการเลือก หรือไม่เลือกของ ชิ้นที่ 1 ถึง i ใส่ถุงเป้ที่รับน้ำหนักได้ไม่เกิน w
 - พิจารณากรณีที่เลือกของชิ้นที่ i และกรณีที่ไมเลือกของชิ้นที่ i ว่ากรณีใด ให้มูลค่ารวมสูงสุดโดยไม่ทำให้น้ำหนักเกิน
 - กรณีไม่เลือกของชิ้นที่ i : เหลือสิ่งของต้องพิจารณา $i - 1$ ชิ้น แต่น้ำหนักที่ ้ถูกเป้รับได้ยังคงเท่าเดิมคือ w ซึ่งจะได้ $V[i, w] = V[i-1, w]$
 - กรณีเลือกของชิ้นที่ i : น้ำหนักที่ถูกรับได้จะลดลงไป w_i ดังนั้นจะได้ $V[i, w] = v_i + V[i-1, w-w_i]$ เมื่อ v_i คือมูลค่าของชิ้นที่ i ที่เพิ่มขึ้น
 - กรณีที่ไม่มีของให้พิจารณา หรือถูกเป้รับน้ำหนักไม่ได้ นั่นคือ $V[0, j] = 0$ และ $V[i, 0] = 0$

Dynamic Programming for Knapsack Problem

- Define an optimal solution to recurrence relation (con't)

$$V[i, w] = \begin{cases} \max(V[i-1, w], v_i + V[i-1, w-w_i]) & \text{if } i > 0 \text{ and } w \geq w_i \\ V[i-1, w] & \text{if } w < w_i \\ 0 & \text{if } i = 0 \text{ or } w = 0 \end{cases}$$

Dynamic Programming for Knapsack Problem

- Decompose a problem into sub-problems.
- Construct an array $V[0..n, 0..w]$ for $1 \leq i \leq n$ and $0 \leq w \leq W$ to keep each computing time of sub-problem.

$V[i, w]$	$w = 0$	1	2	3	...	W
$i = 0$						
1						
2						
...						
n						

Dynamic Programming for Knapsack Problem

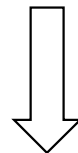
- Define an optimal solution to smaller problem
 - Initial setting:
 - For $0 \leq w \leq W$, set $V[0, w] = 0$
 - For $1 \leq i \leq n$, set $V[i, 0] = 0$

$V[i, w]$	$w = 0$	1	2	3	...	w
$i = 0$	0	0	0	0	...	0
1	0					
2	0					
...	...					
n	0					

Dynamic Programming for Knapsack Problem

- Bottom up computing $V[i, w]$ using iteration (no need recursion)

$V[i, w]$	$w = 0$	1	2	3	...	w
$i = 0$	0	0	0	0	...	0
1	0					
2	0					
...	...					
n	0					



Dynamic Programming for Knapsack Problem

- Algorithm (return maximum value)

```

Initialize all  $V[i, 0]=0$  and all  $V[0, w]=0$ 
for  $i = 1$  to  $n$ 
  for  $w = 1$  to  $W$ 
    if ( $w_i \leq w$ )
       $V[i, w] = \max(V[i-1, w], v_i + V[i-1, w-w_i])$ 
    else
       $V[i, w] = V[i-1, w]$ 
return  $V[n, W]$ 

```

Dynamic Programming for Knapsack Problem

- ตัวอย่าง กำหนดให้ $W = 10$

i	1	2	3	4
v_i	10	40	30	50
w_i	5	4	6	3

```

Initialize all  $V[i, 0]=0$  and all  $V[0, w]=0$ 
for i = 1 to n
  for w = 1 to W
    if ( $w_i \leq w$ )
       $V[i, w] = \max(V[i-1, w], v_i + V[i-1, w-w_i])$ 
    else
       $V[i, w] = V[i-1, w]$ 
return  $V[n, W]$ 

```

$V[i, w]$	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0										
2	0										
3	0										
4	0										

Dynamic Programming for Knapsack Problem

- ตัวอย่าง กำหนดให้ $W = 10$

i	1	2	3	4
v_i	10	40	30	50
w_i	5	4	6	3

```

Initialize all  $V[i, 0]=0$  and all  $V[0, w]=0$ 
for i = 1 to n
  for w = 1 to W
    if ( $w_i \leq w$ )
       $V[i, w] = \max(V[i-1, w], v_i + V[i-1, w-w_i])$ 
    else
       $V[i, w] = V[i-1, w]$ 
return  $V[n, W]$ 

```

$V[i, w]$	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0									
2	0										
3	0										
4	0										

$$V[1, 1] = V[0, 1] = 0$$

$$i = 1 \\ w = 1$$

Dynamic Programming for Knapsack Problem

- ตัวอย่าง กำหนดให้ $W = 10$

i	1	2	3	4
v_i	10	40	30	50
w_i	5	4	6	3

```

Initialize all  $V[i, 0]=0$  and all  $V[0, w]=0$ 
for i = 1 to n
  for w = 1 to W
    if ( $w_i \leq w$ )
       $V[i, w] = \max(V[i-1, w], v_i + V[i-1, w-w_i])$ 
    else
       $V[i, w] = V[i-1, w]$ 
return  $V[n, W]$ 

```

$V[i, w]$	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0								
2	0										
3	0										
4	0										

$$V[1, 2] = V[0, 2] = 0$$

$$i = 1 \\ w = 2$$

Dynamic Programming for Knapsack Problem

- ตัวอย่าง กำหนดให้ $W = 10$

i	1	2	3	4
v_i	10	40	30	50
w_i	5	4	6	3

```

Initialize all  $V[i, 0]=0$  and all  $V[0, w]=0$ 
for i = 1 to n
  for w = 1 to W
    if ( $w_i \leq w$ )
       $V[i, w] = \max(V[i-1, w], v_i + V[i-1, w-w_i])$ 
    else
       $V[i, w] = V[i-1, w]$ 
return  $V[n, W]$ 

```

$V[i, w]$	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	10					
2	0										
3	0										
4	0										

$$\text{Max}(V[0, 5], 10 + V[0, 0])$$

$$i = 1 \\ w = 5 \\ \text{Max}(0, 10)$$

Dynamic Programming for Knapsack Problem

- ตัวอย่าง กำหนดให้ $W = 10$

i	1	2	3	4
v_i	10	40	30	50
w_i	5	4	6	3

```

Initialize all  $V[i, 0]=0$  and all  $V[0, w]=0$ 
for i = 1 to n
  for w = 1 to W
    if ( $w_i \leq w$ )
       $V[i, w] = \max(V[i-1, w], v_i + V[i-1, w-w_i])$ 
    else
       $V[i, w] = V[i-1, w]$ 
return  $V[n, W]$ 

```

$V[i, w]$	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	10	10	10	10	10	10
2	0										
3	0										
4	0										

Max($V[0,10]$,
 $10+V[0,5]$)

$i = 1$
 $w = 10$
Max($0, \underline{10}$)

Dynamic Programming for Knapsack Problem

- ตัวอย่าง กำหนดให้ $W = 10$

i	1	2	3	4
v_i	10	40	30	50
w_i	5	4	6	3

```

Initialize all  $V[i, 0]=0$  and all  $V[0, w]=0$ 
for i = 1 to n
  for w = 1 to W
    if ( $w_i \leq w$ )
       $V[i, w] = \max(V[i-1, w], v_i + V[i-1, w-w_i])$ 
    else
       $V[i, w] = V[i-1, w]$ 
return  $V[n, W]$ 

```

$V[i, w]$	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	10	10	10	10	10	10
2	0	0									
3	0										
4	0										

$V[2,1] = V[1,1]$
 $= 0$

$i = 2$
 $w = 1$

Dynamic Programming for Knapsack Problem

- ตัวอย่าง กำหนดให้ $W = 10$

i	1	2	3	4
v_i	10	40	30	50
w_i	5	4	6	3

```

Initialize all  $V[i, 0]=0$  and all  $V[0, w]=0$ 
for i = 1 to n
  for w = 1 to W
    if ( $w_i \leq w$ )
       $V[i, w] = \max(V[i-1, w], v_i + V[i-1, w-w_i])$ 
    else
       $V[i, w] = V[i-1, w]$ 
return  $V[n, W]$ 

```

$V[i, w]$	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	10	10	10	10	10	10
2	0	0	0	0	40						
3	0										
4	0										

Max($V[1,4]$,
 $40+V[1,0]$)

$i = 2$
 $w = 4$
Max($0, \underline{40}$)

Dynamic Programming for Knapsack Problem

- ตัวอย่าง กำหนดให้ $W = 10$

i	1	2	3	4
v_i	10	40	30	50
w_i	5	4	6	3

```

Initialize all  $V[i, 0]=0$  and all  $V[0, w]=0$ 
for i = 1 to n
  for w = 1 to W
    if ( $w_i \leq w$ )
       $V[i, w] = \max(V[i-1, w], v_i + V[i-1, w-w_i])$ 
    else
       $V[i, w] = V[i-1, w]$ 
return  $V[n, W]$ 

```

$V[i, w]$	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	10	10	10	10	10	10
2	0	0	0	0	40	40	40	40	40	50	
3	0										
4	0										

Max($V[1,9]$,
 $40+V[1,5]$)

$i = 2$
 $w = 9$
Max($10, \underline{50}$)

Dynamic Programming for Knapsack Problem

- ตัวอย่าง กำหนดให้ $W = 10$

i	1	2	3	4
v_i	10	40	30	50
w_i	5	4	6	3

$V[i, w]$	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	10	10	10	10	10	10
2	0	0	0	0	40	40	40	40	40	50	50
3	0	0	0	0	40	40	40	40	40	50	70
4	0	0	0	50	50	50	50	90	90	90	90

Dynamic Programming for Knapsack Problem

- How to know which items included in knapsack?

```

Initialize all  $V[i, 0]=0$  and all  $V[0, w]=0$ 
for i = 1 to n
  for w = 0 to W
    if ( $w_i \leq w$ ) and ( $v_i + V[i-1, w-w_i] > V[i-1, w]$ )
       $V[i, w] = v_i + V[i-1, w-w_i]$ 
      item[i,w]=1 //include
    else
       $V[i, w] = V[i-1, w]$ 
      item[i,w]=0 //not include
  //continue ->

```

Dynamic Programming for Knapsack Problem

- How to know which items included in knapsack? (con't)

```

//... print items that choose into a knapsack
x=W
For i = n downto 1
  if (item[i, x]==1)
    print i
    x = x -  $w_i$ 
return  $V[n, W]$ 

```

Problem (knapsack_1)

มานิอยากทำอาหารให้มานะทาน จึงเดินทางไปจ่ายตลาดด้วยตัวเอง และด้วยความที่เป็นผู้หญิงจึงต้องการเลือกวัตถุดิบที่ให้คุณค่าทางโภชนาการสูงสุดและมีความหลากหลาย โดยที่น้ำหนักรวมของวัตถุดิบต้องไม่เกินกำลังที่มานิจะถือกลับได้ จึงเขียนโปรแกรมเพื่อช่วยมานิซื้อวัตถุดิบ

Input:

บรรทัดที่ 1 คือ จำนวนชนิดของวัตถุดิบในตลาด (n) และน้ำหนัก รวมที่มานิสามารถถือได้ (W)

บรรทัดที่ 2 คือ น้ำหนักของวัตถุดิบแต่ละชนิด ($w_1 w_2 w_3 \dots w_n$)

บรรทัดที่ 3 คือ คุณค่าทางโภชนาการของวัตถุดิบแต่ละชนิด ($v_1 v_2 v_3 \dots v_n$)

Problem (ต่อ)

Output: คุณค่าทางโภชนาการรวมสูงสุด จำนวนชนิดของวัตถุดิบที่เลือก
และน้ำหนักรวมที่مانีต้องถือของกลับ

Sample

Input	Output
4 10 6 3 4 2 30 14 16 9	46 2 10
5 60 20 10 40 10 30 40 100 50 60 30	210 3 60

MATRIX CHAIN MULTIPLICATION

การคูณเมทริกซ์

ถ้า $A = [a_{ij}]_{m \times n}$ และ $B = [b_{ij}]_{p \times q}$



$A \times B$ หาค่าได้เมื่อ $n = p$

$A \times B = C$ โดยที่ C มีมิติ $m \times q$

ซึ่ง $C = [c_{ij}]_{m \times q}$ โดยที่

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \dots + a_{in}b_{nj}$$

EX ตัวอย่างการคูณเมทริกซ์

$$\text{ให้ } A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}_{2 \times 3} \text{ และ } B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix}_{3 \times 2}$$

$$\begin{aligned} A \times B &= \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}_{2 \times 3} \times \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix}_{3 \times 2} \\ &= \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} \end{bmatrix} \\ &= \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}_{2 \times 2} \end{aligned}$$

Matrix Chain Multiplication

- กำหนดให้มีเมทริกซ์จำนวน n เมทริกซ์ A_1, A_2, \dots, A_n
ผลคูณลูกโซ่เมทริกซ์ (Matrix chain multiplication) คือ

$$A_1 \times A_2 \times \dots \times A_n$$

- โดย
 - $A \times B \neq B \times A$
 - $A \times (B \times C) = (A \times B) \times C$

Matrix Chain Multiplication

- การหาผลคูณของเมทริกซ์ $A_1 \times A_2$ ที่มีขนาด $p_0 \times p_1$ และ $p_1 \times p_2$ ใช้เวลาเท่ากับ $p_0 p_1 p_2$ ครั้ง
- ตัวอย่าง สมมติให้ A_1, A_2, A_3 เป็นเมทริกซ์ขนาด 10×100 , 100×5 และ 5×50 ตามลำดับ
 - $(A_1 \times A_2) \times A_3$ ใช้เวลาเท่ากับ

$$(10 \times 100 \times 5) + (10 \times 5 \times 50) = 5000 + 2500 = 7,500$$
 - $A_1 \times (A_2 \times A_3)$ ใช้เวลาเท่ากับ

$$(10 \times 100 \times 50) + (100 \times 5 \times 50) = 50000 + 25000 = 75,000$$

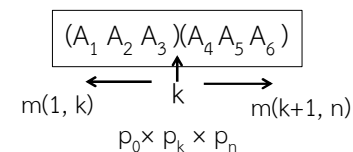
Dynamic Programming for Matrix Chain Multiplication

- Decompose a problem into sub-problems
 - กำหนดให้ A_1, A_2, \dots, A_n คือ matrix chain
 - A_i มีขนาดเท่ากับ $p_{i-1} \times p_i$
 - A_{ij} แทนผลลัพธ์ที่ได้จาก $A_i \times A_{i+1} \times \dots \times A_j$ ดังนั้น
 - A_{ij} มีขนาดเท่ากับ $p_{i-1} \times p_j$
 - การคูณ $(A_{ij})(A_{j+1,k})$ ใช้จำนวนการคูณเท่ากับ $p_{i-1} \times p_j \times p_k$ ครั้ง
 - กำหนดให้ $m[i, j]$ เก็บจำนวนการคูณที่น้อยที่สุดของการคูณ $A_i \times A_{i+1} \times \dots \times A_j$

Dynamic Programming for Matrix Chain Multiplication

- Decompose a problem into sub-problems (con't)
 - แบ่ง matrix chain เป็น 2 ส่วน ที่ตำแหน่งที่ k ที่ทำให้ได้จำนวนการคูณที่น้อยที่สุด นั่นคือ

$$m(1, k) + m(k+1, n) + p_0 \times p_k \times p_n$$
 หมายเหตุ ผลรวมนี้น้อยสุดเมื่อ $m(1, k)$ และ $m(k+1, n)$ น้อยที่สุด



Dynamic Programming for Matrix Chain Multiplication

- Define an optimal solution to recurrence relation

$$m(i, j) = \begin{cases} \min_{i \leq k < j} (m(i, k) + m(k+1, j) + p_{i-1}p_kp_j) & \text{if } i < j \\ 0 & \text{if } i = j \end{cases}$$

Dynamic Programming for Matrix Chain Multiplication

- Define an optimal solution to smaller problem
 - Initial setting:
 - $m(i, j) = 0$ ถ้า $i = j$

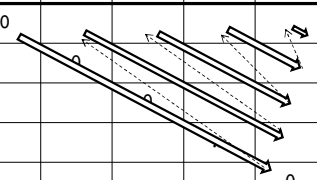
m(i, j)	j = 1	2	3	...	n
i = 1	0				
2		0			
3			0		
...				...	
n					0

Dynamic Programming for Matrix Chain Multiplication

- Bottom up computing

$$m(i, j) = \min_{i \leq k < j} (m(i, k) + m(k+1, j) + p_{i-1}p_kp_j)$$

m[i, j]	j = 1	2	3	...	n
i = 1	0				
2					
3					
...					
n					0



Dynamic Programming for Matrix Chain Multiplication

- Algorithm (return the lowest number of multiplication)

```

matrixChain_DP (p[0..n])
  for i = 0 to n //initial setting
    m[i, i] = 0
  for len = 2 to n //diagonals
    for i = 1 to n-len+1 // rows with an entry on d-th diagonal
      j = i + len - 1 //column corresp. to row i on d-th diagonal
      m[i, j] = INT_MAX //infinity
      for k = i to j-1
        q = m[i, k] + m[k+1, j] + p[i-1]*p[k]*p[j]
        if (q < m[i, j])
          m[i, j] = q
  return m[1, n]
  
```

Dynamic Programming for Matrix Chain Multiplication

- ตัวอย่าง ต้องการหาวิธี matrix chain multiplication ของ $A_1 A_2 A_3 A_4 A_5 A_6$ ที่มีขนาด $30 \times 20, 20 \times 10, 10 \times 5, 5 \times 5, 5 \times 10$ และ 10×40 ตามลำดับ
- จะได้ $p[0..6] = \{30, 20, 10, 5, 5, 10, 40\}$

Dynamic Programming for Matrix Chain Multiplication

- ตัวอย่าง (ต่อ) $p[0..6] = \{30, 20, 10, 5, 5, 10, 40\}$

m[i, j]	1	2	3	4	5	6
1	0					
2		0				
3			0			
				0		
					0	
						0

```

matrixChain_DP (p[0..n])
for i = 0 to n
    m[i, i] = 0
for len = 2 to n //diagonals
    for i = 1 to n-len+1
        j = i + len - 1
        m[i, j] = INT_MAX
        for k = i to j-1
            q = m[i, k] + m[k+1, j] + p[i-1]*p[k]*p[j]
            if (q < m[i, j])
                m[i, j] = q
return m[1, n]

```

Dynamic Programming for Matrix Chain Multiplication

- ตัวอย่าง (ต่อ) $p[0..6] = \{30, 20, 10, 5, 5, 10, 40\}$

```

len = 2
i = 1
j = 2
k = 1
q = 0+0+30*20*10

```

m[i, j]	1	2	3	4	5	6
1	0	6000				
2		0				
3			0			
				0		
					0	
						0

```

matrixChain_DP (p[0..n])
for i = 0 to n
    m[i, i] = 0
for len = 2 to n //diagonals
    for i = 1 to n-len+1
        j = i + len - 1
        m[i, j] = INT_MAX
        for k = i to j-1
            q = m[i, k] + m[k+1, j] + p[i-1]*p[k]*p[j]
            if (q < m[i, j])
                m[i, j] = q
return m[1, n]

```

$$6 - 2 + 1 = 5$$

$$2 - 1 = 1$$

$$m[1,1] + m[2,2] + p_0 * p_1 * p_2$$

Dynamic Programming for Matrix Chain Multiplication

- ตัวอย่าง (ต่อ) $p[0..6] = \{30, 20, 10, 5, 5, 10, 40\}$

```

len = 2
i = 2
j = 3
k = 2
q = 0+0+20*10*5

```

m[i, j]	1	2	3	4	5	6
1	0	6000				
2		0	1000			
3			0			
				0		
					0	
						0

```

matrixChain_DP (p[0..n])
for i = 0 to n
    m[i, i] = 0
for len = 2 to n //diagonals
    for i = 1 to n-len+1
        j = i + len - 1
        m[i, j] = INT_MAX
        for k = i to j-1
            q = m[i, k] + m[k+1, j] + p[i-1]*p[k]*p[j]
            if (q < m[i, j])
                m[i, j] = q
return m[1, n]

```

$$6 - 2 + 1 = 5$$

$$3 - 1 = 2$$

$$m[2,2] + m[3,3] + p_1 * p_2 * p_3$$

Dynamic Programming for Matrix Chain Multiplication

- ตัวอย่าง (ต่อ) $p[0..6] = \{30, 20, 10, 5, 5, 10, 40\}$

len = 2
i = 3, 4, 5

m[i, j]	1	2	3	4	5	6
1	0	6000				
2		0	1000			
3			0	250		
4				0	250	
5					0	2000
6						0

```
matrixChain_DP (p[0..n])
for i = 0 to n
  m[i,i] = 0
for len = 2 to n //diagonals
  for i = 1 to n-len+1
    j = i + len - 1
    m[i, j] = INT_MAX
    for k = i to j-1
      q = m[i, k] + m[k+1, j] + p[i-1]*p[k]*p[j]
      if (q < m[i, j])
        m[i, j] = q
return m[1, n]
```

$$6 - 2 + 1 = 5$$

Dynamic Programming for Matrix Chain Multiplication

- ตัวอย่าง (ต่อ) $p[0..6] = \{30, 20, 10, 5, 5, 10, 40\}$

len = 3
i = 1
j = 3
k = 1
q = 0+1000+30*20*5

m[i, j]	1	2	3	4	5	6
1	0	6000	4000			
2		0	1000			
3			0	250		
4				0	250	
5					0	2000
6						0

```
matrixChain_DP (p[0..n])
for i = 0 to n
  m[i,i] = 0
for len = 2 to n //diagonals
  for i = 1 to n-len+1
    j = i + len - 1
    m[i, j] = INT_MAX
    for k = i to j-1
      q = m[i, k] + m[k+1, j] + p[i-1]*p[k]*p[j]
      if (q < m[i, j])
        m[i, j] = q
return m[1, n]
```

$$6 - 3 + 1 = 4$$

$$3 - 1 = 2$$

$$m[0,0] + m[2,3] + p_0*p_1*p_3$$

Dynamic Programming for Matrix Chain Multiplication

- ตัวอย่าง (ต่อ) $p[0..6] = \{30, 20, 10, 5, 5, 10, 40\}$

len = 3
i = 1
j = 3
k = 2
q = 6000+0+30*10*5

m[i, j]	1	2	3	4	5	6
1	0	6000	4000			
2		0	1000			
3			0	250		
4				0	250	
5					0	2000
6						0

```
matrixChain_DP (p[0..n])
for i = 0 to n
  m[i,i] = 0
for len = 2 to n //diagonals
  for i = 1 to n-len+1
    j = i + len - 1
    m[i, j] = INT_MAX
    for k = i to j-1
      q = m[i, k] + m[k+1, j] + p[i-1]*p[k]*p[j]
      if (q < m[i, j])
        m[i, j] = q
return m[1, n]
```

$$6 - 3 + 1 = 4$$

$$3 - 1 = 2$$

$$m[1,2] + m[3,3] + p_0*p_2*p_3$$

$$7500 > 4000$$

Dynamic Programming for Matrix Chain Multiplication

- ตัวอย่าง (ต่อ) $p[0..6] = \{30, 20, 10, 5, 5, 10, 40\}$

len = 3
i = 2
j = 4
k = 2
q = 0+250+20*10*5

m[i, j]	1	2	3	4	5	6
1	0	6000	4000			
2		0	1000	1250		
3			0	250		
4				0	250	
5					0	2000
6						0

```
matrixChain_DP (p[0..n])
for i = 0 to n
  m[i,i] = 0
for len = 2 to n //diagonals
  for i = 1 to n-len+1
    j = i + len - 1
    m[i, j] = INT_MAX
    for k = i to j-1
      q = m[i, k] + m[k+1, j] + p[i-1]*p[k]*p[j]
      if (q < m[i, j])
        m[i, j] = q
return m[1, n]
```

$$6 - 3 + 1 = 4$$

$$4 - 1 = 3$$

$$m[2,2] + m[3,4] + p_1*p_2*p_4$$

Dynamic Programming for Matrix Chain Multiplication

- ตัวอย่าง (ต่อ) $p[0..6] = \{30, 20, 10, 5, 5, 10, 40\}$

```
len = 3
i = 2
j = 4
k = 3
q = 1000+0+20*5*5
```

m[i, j]	1	2	3	4	5	6
1	0	6000	4000		1500 > 1250	
2		0	1000	1250		
3			0	250		
4				0	250	
5					0	2000
6						0

```
matrixChain_DP (p[0..n])
for i = 0 to n
  m[i,i] = 0
for len = 2 to n //diagonals
  for i = 1 to n-len+1
    j = i + len - 1
    m[i, j] = INT_MAX
    for k = i to j-1
      q = m[i, k] + m[k+1, j] + p[i-1]*p[k]*p[j]
      if (q < m[i, j])
        m[i, j] = q
return m[1, n]
```

$$6 - 3 + 1 = 4$$

$$4 - 1 = 3$$

$$m[2,3] + m[4,4] + p_1*p_3*p_4$$

Dynamic Programming for Matrix Chain Multiplication

- ตัวอย่าง (ต่อ) $p[0..6] = \{30, 20, 10, 5, 5, 10, 40\}$

```
len = 3
i = 3, 4
```

m[i, j]	1	2	3	4	5	6
1	0	6000	4000			
2		0	1000	1250		
3			0	250	2250	
4				0	250	4250
5					0	2000
6						0

```
matrixChain_DP (p[0..n])
for i = 0 to n
  m[i,i] = 0
for len = 2 to n //diagonals
  for i = 1 to n-len+1
    j = i + len - 1
    m[i, j] = INT_MAX
    for k = i to j-1
      q = m[i, k] + m[k+1, j] + p[i-1]*p[k]*p[j]
      if (q < m[i, j])
        m[i, j] = q
return m[1, n]
```

Dynamic Programming for Matrix Chain Multiplication

- ตัวอย่าง (ต่อ) $p[0..6] = \{30, 20, 10, 5, 5, 10, 40\}$

m[i, j]	1	2	3	4	5	6
1	0	6000	4000	4250	5750	12250
2		0	1000	1250	2250	7250
3			0	250	750	4250
4				0	250	2250
5					0	2000
6						0

Dynamic Programming for Matrix Chain Multiplication

- How to find the position of separated matrix?

```
matrixChain_DP (p[0..n])
for i = 0 to n
  m[i,i] = 0
for len = 2 to n //diagonals
  for i = 1 to n-len+1
    j = i + len - 1
    m[i, j] = INT_MAX
    for k = i to j-1
      q = m[i, k] + m[k+1, j] + p[i-1]*p[k]*p[j]
      if (q < m[i, j])
        m[i, j] = q
        s[i, j] = k
return s
```

Dynamic Programming for Matrix Chain Multiplication

- ตัวอย่าง (ต่อ) $p[0..6] = \{30, 20, 10, 5, 5, 10, 40\}$

$s[i, j]$	1	2	3	4	5	6
1	0	1	1	1	3	3
2		0	2	2	3	3
3			0	3	3	3
4				0	4	5
5					0	5
6						0

$s[1, 6] = 3 \rightarrow (A_1 A_2 A_3) (A_4 A_5 A_6)$

$s[1, 3] = 1 \rightarrow (A_1(A_2 A_3)) (A_4 A_5 A_6)$

$s[4, 6] = 5 \rightarrow (A_1(A_2 A_3))(A_4 A_5 A_6)$

Problem (mChain_1)

จงเขียนโปรแกรมเพื่อหาจำนวนการคูณกันน้อยที่สุดของ matrix chain ที่กำหนดให้

Input: บรรทัดที่ 1 คือ จำนวนเมทริกซ์ n

บรรทัดที่ 2 คือ ขนาดของเมทริกซ์ $p_0 p_1 p_2 p_3 \dots p_n$

Output: จำนวนการคูณที่น้อยที่สุด

Sample

Input	Output
6 30 20 10 5 5 10 40	12250
3 10 100 5 50	7500